Reinforcement Learning: An Introduction
Summary Notes

Mrinank Sharma

March 30, 2020

1 Introduction

Elements of Reinforcement Learning

Other than the **agent** and the **environment**, there are four main subelements of an RL system:

1. The *policy* maps perceived states of the environment to actions. This could be a lookup table or a search process.
2. The *reward signal* defines the goal of a reinforement learning problem. The environment sends rewards to an agent that has the sole objective of maximising this reward.
3. The *value function* specifies what is good in the long run. The *value* of a state is the total amount of reward an agent can expect to accumulate over teh future starting from that state.
4. Some RL systems have *models* of the environment that can mimic the behaviour of the environment. Models are used for planning.

Rewards and Value Functions

Whilst rewards determine the immediate, intrisic desirability of environmental states, values indicate the long-term desirability of states after taking into account which states are likely to follow. We choose actions based on values, even though values are derived from rewards.

Evolutionary Methods

It is possible to learn policies without estimating value functions by using evolutionary methods. However, they tend to ignore much of the useful structure and are not well suited for RL tasks as they ignore relevant information, such as the states which are passed and the actions which are selected.

2 Multi-armed Bandits

> *The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that* ***evaluates*** *the actions taken rather than* ***instructs*** *by giving correct actions.*

This form of feedback creates the need for active exploration.

k-armed bandits

Definition 2.1 (k-armed Bandit) *In the k-armed bandit problem, you are repeatedly faced with a choice among k different actions. After each choice, a numerical reward is received from a* ***stationary*** *probability distribution associated with the selected action. The objective is to maximise the expected total reward over some time period.*

Value

The value of action a is the expected reward given a was chosen:

$$q_*(a) = \mathbb{E}[R_t \,|\, A_t = a]. \tag{1}$$

If we knew the values, problem solved - just pick the action with the maximum value. However, we do not know these. Denote the *estimated* value as $Q_t(a)$. A simple way to estimate the action-value is a *sample average*, which can be calculated incrementally.

Reward Variance

Note that purely greedy methods are best if there is no noise in the reward signal. With ϵ-greedy algorithms, the best value of ϵ will depend on the amount of noise, which determines the number of samples for the sample average to converge. However, if the problem is *non-stationary*, we still need to explore.

Non-stationary Problems

If the bandit is non-stationary, we might use the following rule for value estimates:

$$Q_{n+1}(a) = Q_n(a) + \alpha_n(a)\left[R_n - Q_n(a)\right]. \tag{2}$$

Convergence Guarantees

It is known that convergence with probability 1 happens if the following conditions hold:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty, \text{ and } \sum_{n=1}^{\infty} \alpha_n(a)^2 < \infty. \tag{3}$$

The first condition ensures the steps are sufficiently large to overcome an initial condition and the second condition ensures that the steps become small enough to give convergence. However, step sizes which meet this conditions are rarely used in applications because they seem to converge slowly or require considerable tuning.

With this techniques, the initialisation effectively becomes an additional parameter which needs to be chosen. They can be used to provide prior knowledge and can encourage exploration, for example, by being optimistic. However, this sort of exploration is not useful for non-stationary problems (the task changes creates a renewed need for exploration).

UCB

The intuition behind UCB is that its better, when exploring, to select among the non-greedy actions which have the *potential* for being optimal. To do this, we maximise an estimate of the arm's value summed with some sort of uncertainty estimate. UCB often performs well but can be difficult to extent to RL from bandits.

We don't have to use value based methods. Instead, we could directly learn a preference over the actions and modify our preferences to give us higher rewards e.g., increasing our preference for an action if selecting it lead to a better outcome than expected.

Contextual Bandits

Definition 2.2 (Contextual Bandits) *In a contextual bandit, the agent is faced with a series of multi-armed bandit problems, each associated with some information i.e., the agent has information about which bandit problem is being faced at any timestep.*

Contextual bandits are intermediate between multi-armed bandits and the full RL problem; they involve learning a policy (i.e., a mapping from state information to actions) but similar to k-armed bandits, **the action selected influences only the immediate reward and not the next situation**. If we drop this constraint, we have the full RL problem.

3 Finite Markov Decision Processes

MDP

Definition 3.1 (Finite Markov Decision Process (MDP)) *A finite Markov Decision Process (MDP) is made up of:*

- *A sequence of discrete time steps,* $t = 0, 1, 2, \ldots$.
- *A discrete set of states,* $S_t \in \mathcal{S}$.
- *A set of actions which the agent can take,* $A_t \in \mathcal{A}(S_t)$.
- *A finite set of numerical rewards,* $R_t \in \mathcal{R} \subset \mathbb{R}$. *In this case, there is a well defined discrete transition distribution:*

$$p(s', r|s, a) \triangleq Pr[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a], \tag{4}$$

which defines the ***dynamics*** *of the MDP.*

Interpreting the Markov Assumption

Note the Markov assumption; probabilities over the next state and reward depend only on the current state and action. This can be interpreted as a restriction on the state - the state must include information about all aspects of the past agent-environment interaction which will affect the future.

Given p, we can compute the following functions of interest:

$$p(s'|s, a) \triangleq \Pr[S_t = s'|S_{t-1} = s, A_{t-1} = a], \tag{5}$$

$$r(s, a) \triangleq \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a], \tag{6}$$

and

$$r(s,a,s') \triangleq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s']. \tag{7}$$

Agent vs Environment

A general rule of thumb is that anything that cannot be changed *arbitarily* by the agent is considered to be outside of it and thus part of the environment. The agent-environment boundary represents the limit of the agent's absolute control, not of its knowledge.

The goal of the agent is formalised in terms of the reward signal which passes from the environment to the agent. This is one of the most distinctive features of RL.

Reward Hypothesis

> *All of what we mean by goals and purposes can be well though of as the maximisation of the expected value of the cumulative sum of a received scalar signal.*

The reward signal is not the place to impart to the agent prior knowlege about *how* to achieve a task; if we do this, the agent may only achieve subgoals without actually acheiving the real goal. A better place to impart this knowledge is the initial policy or value function. **How does this link to reward shaping?**

Discount Rates

Episodic tasks terminate at some timestep, T, whilst continuing tasks do not. In the first case, we simply use the (expected) sum of rewards whilst we introduce discounting in the infinite horizon case i.e.,

$$G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{8}$$

G_t is the *discounted return*. The closer the value of γ is to 1, the more farsighted the agent is.

Unifying Notation

We convert episodic tasks to continuing tasks by introducting a special *absorbing state* which transitions only to itself and returning zero reward. We can write:

$$G_t \triangleq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{9}$$

allowing for $T = \infty$ and $\gamma = 1$. **The book says not both - but I think that should be allowed for an episodic task with absorbing state.**

Policy

Definition 3.2 (Policy) *A policy is a mapping from states to probabilities of selecting each possible action. $\pi(a|s)$ is the distribution over actions, A_t, given the current state, S_t.*

Value Function

Definition 3.3 (Value Function) *The value function of state s under policy π is the expected return when starting in s and following π.*

$$v_\pi(s) \triangleq \mathbb{E}_\pi[G_t | S_t = s] \tag{10}$$

Action-Value Function

Definition 3.4 (Action-Value Function) *The action-value function of action a, state s under policy π is the expected return when starting in s and following π after first performing a.*

$$q_\pi(s,a) \triangleq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{11}$$

Value functions satisfy recursive relationships.

Optimal Policies

A policy, π, is better than policy π' iff $v_\pi(s) \geq v_{\pi'}(s) \forall s \in \mathcal{S}$ with a strict inequality for at least one state. Optimal policies are better than or equal to all other policies. There may be multiple optimal policies, but they all share the same state-value function,

$$v_*(s) = \max_\pi v_\pi(s), \text{ for all } s \in \mathcal{S}. \tag{12}$$

The optimal action-value function is define analogously.

Bellman Optimality Equality

The Bellman optimality equation for the value function is:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a)[r + \gamma v_*(s')], \tag{13}$$

which expresses the intuition that the optimal value of a state is equal to the expected return of the best action from that state. Similarly, for the action-value function,

$$q_*(s,a) = \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a' \in \mathcal{A}(s')} q_*(s', a')]. \tag{14}$$

For a finite MDP, the Bellman optimality equations give a system of equations which can be solved for $v_*(s)$. Once we have the optimal value functions, it is easy to determine an optimal policy by acting greedily (or performing a one-step search). **If you use v_* to select actions in the short-term, the greedy policy is optimal in the long term since this value function takes into account reward consequences.** The optimal expected long-term return is locally and immediately available for each state so a one-step lookahead search yields optimal actions.

Note that the action-value function effectively caches the results of all of the one-step lookahead searches. This function can be maximised (with respect to the action) to find the optimal policy.

Caveats

This all sounds great. However, we run into the following problems:

- We need to accurately know the dynamics of the environment.
- We need to have enough computational resources.
- The Markov property needs to hold.

4 Dynamic Programming

Limited Utility

Dynamic Programming (DP) algorithms can be used to compute optimal policies given perfect models of the environment. However, they are important only theoretically because they are limited by imperfect models and finite compute.

Policy Evaluation

Policy evaluation computes the state-value function for policy π. We turn the Bellman equation into an update rule:

$$v_{k+1}(s) \triangleq \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s, a)[r + \gamma v_k(s)], \quad \forall s \in \mathcal{S}. \tag{15}$$

v_π is a fixed point for this update rule since it must satisfy the Bellman equation. Note that the estimate in general converges as $k \to \infty$ provided $\gamma < 1$ or eventual termination from all states under π (the same conditions which give a unique value function).

This is an *expected update*; the old value is replaced with a new value computed using previous values and expected immediate reward. In fact, all DP updates are called *expected updates* since they take the expectation over successor states rather than sampling.

Algorithm 1 Iterative Policy Evaluation

Input: π, the policy to be evaluated
Parameters: $\theta > 0$, a threshold determining accuracy.
Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

```
loop:
    Δ ← 0
    for each s ∈ S do:        ▷ Sweep over states, using the most recent values of V(s)
        v ← V(s)
        V(s) ← Σ_{a∈A(s)} π(a|s) Σ_{s'∈S,r∈R} p(s',r|s,a)[r + γV(s')]
        Δ ← max(Δ, |v − V(s)|)
until Δ < θ
```

If we know the dynamics, knowing the value of the policy helps us improve the policy. We can compute the value of selecting action a in state s and thereafter following π. If it is better to select a once in s and thereafter follow π, it is better to always select a in s and the new policy is an improvement.

Policy Improvement Theorem

Theorem 4.1 (Policy Improvement Theorem) *Let π and π' be any pair of determinstic policies such that, for all $s \in \mathcal{S}$,*

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{16}$$

Then π' must be as good as, or better than, π i.e., it must obtain greater or equal expected return from all states $s \in \mathcal{S}$:

$$v'_\pi(s) \geq v_\pi(s). \tag{17}$$

Note that if there is any strict inequality in Eq. (16)*, there must be a strict inequality in Eq.* (17) *in* at least *one state.*

Policy Improvement

The greedy policy i.e., $\pi'(s) = \arg\max_a q_\pi(s,a)$, meets the conditions of the policy improvement theorem. The process of making a new policy which improves on an original policy by making it greedy with respect its value function is known as *policy improvement*. If policy improvement gives us a policy with the same perform as the old policy, this policy must necessarily be optimal as only the optimal policy satifies the Bellman optimality equation.

Note that similar analysis can be performed for stochastic policies.

Policy Iteration

We can repeatedly evaluate policy π to calculate v_π and then improve π to form π'. This process is guaranteed to find the optimal policy.

Algorithm 2 Policy Iteration

Initialisation: initialise $V(s)$ arbitarily for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$. Initalise $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$.

Parameters: $\theta > 0$, a threshold determining accuracy.

loop: ▷ Policy Evaluation
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s,a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

`policy-stable` ← True ▷ Policy Improvement
for each $s \in \mathcal{S}$ **do**:
    `old-action` ← $\pi(s)$
    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r|s,a)[r + \gamma V(s')]$
    if `old-action` $\neq \pi(s)$ **then**:
        `policy-stable` ← True
if `policy-stable` **then**:
    return $V \simeq v_*, \pi \simeq \pi*$
else:
    go to 1